Ethernet/IP™ for SoftPLC® Runtime

Version 1.6

Table of Contents

Chapter 1. Terms of Use

Because of the variety of uses of the information described in this manual, the users of, and those responsible for applying this information must satisfy themselves as to the acceptability of each application and use of the information. In no event will SoftPLC Corporation be responsible or liable for its use, nor for any infringements of patents or other rights of third parties which may result from its use.

SOFTPLC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

SoftPLC Corporation reserves the right to change product specifications at any time without notice. No part of this document may be reproduced by any means, nor translated, nor transmitted to any magnetic medium without the written consent of SoftPLC Corporation.

SoftPLC, and TOPDOC are registered trademarks of SoftPLC Corporation.

© Copyright 2018 SoftPLC Corporation, ALL RIGHTS RESERVED Ethernet/IP is a trademark and ODVA is a registered trademark of ODVA, Inc. Rockwell Automation, Allen-Bradley, and ControlLogix, are registered trademarks and CompactLogix is a trademark of Rockwell Automation.
© Copyright 2022 SoftPLC Corporation, ALL RIGHTS RESERVED

First Printing October, 2018

Latest Printing February, 2024

SoftPLC Corporation
25603 Red Brangus Drive
Spicewood, Texas 78669

USA Telephone: 1-800-SoftPLC
WW Telephone: 512/264-8390
URL: http://softplc.com
Email: support@softplc.com

Chapter 2. Overview

2.1. Introduction

The SoftPLC runtime is control software developed by SoftPLC Corporation. It is is embedded into all SoftPLC controllers and gateways. This document describes how to configure and use the Ethernet/IP add-on module for the SoftPLC runtime. This module enables the controller to be both an originator (master/scanner) and a target (slave/adapter) on one Ethernet/IP network.

2.2. Concepts

The SoftPLC runtime engine software supports TLM's (TOPDOC Loadable Modules), which are extensions to SoftPLC. A TLM may be loaded either as a DRIVER or as a MODULE. The difference between a DRIVER and a MODULE is that a DRIVER is called once per SoftPLC scan cycle, and optionally an additional number of times per scan. A MODULE is only called when the control program decides to call it and not as an inherent part of the scan. TLM's are made known to SoftPLC in a MODULES.LST file which may be edited by TOPDOC NexGen by traversing to: **PLC › Modules**.

SoftPLC's Ethernet/IP communications are implemented using a TLM called ETHER_IP, which is a DRIVER and has a number of TLI's (TOPDOC Loadable Instructions) contained within it. TLI's are Ladder Logic instructions that can be used to control or query the operation of the TLM. The DRIVER uses a text configuration file called ETHER_IP.LST to identify the desired Ethernet/IP communications for user applications.

2.3. Features

- Five milli-second timing granularity.
- I/O scanner: up to 128 total originating i/o connections to as many target slaves as needed.
    - each i/o connection may have its own forward open timeout, RPI timeout multiplier, connection type, connection trigger type
    - each i/o connection half may have its own RPI, size, target and originating assembly, and realtime format.
- Up to 20 target i/o connections in the form of exclusive owner
- Up to 3 input only i/o connections
- Up to 2 listen only i/o connections
- I/O connections may be unicast or multicast.
- Explicit messaging server supporting:
    - assembly class 0x04 instances
    - CIP symbol class 0x6b support, per Rockwell Automation publication 756-PM020D-EN-P - June 2016. This lets a SoftPLC runtime look like an Allen-Bradley PLC to various HMIs and to other Allen-Bradley PLCs. A SoftPLC descriptor tag is exported as a 0x6b CIP symbol when

an explicit assembly is created. Currently 16-bit integer blocks and 32-bit float blocks are supported in these class 0x6b explicit assemblies. See Configuring Logix to Communicate with SoftPLC Using Messaging.

- Up to 1000 elements per assembly
- Data values can be 16 bit integer or 32 bit floating point numbers

2.4. Limitations

- No explicit messaging client at this time, only server support as stated in Features above.
- The total number of I/O connection bytes supported is limited by the size of the SoftPLC runtime license. (Contact info@softplc.com for details.)
- One ethernet interface is supported.

2.5. Requirements

- SoftPLC version 4.6 or greater
- TOPDOC NexGen version 1.6 or greater
- Working knowledge of Ethernet/IP concepts
- Vendor information on the Ethernet/IP configurations for devices to be connected to the SoftPLC (PLC's, Drives, HMI's, I/O, etc.)

2.6. Terminology

Ethernet/IP terminology is often confusing, because different terms are often used for the same purpose. For purposes of this document, the following language is used:

- Originator – An Ethernet/IP capable device that initiates the communication. Also known as a Scanner or Master.
- Target – An Ethernet/IP capable device that responds to communication requests from an Originator. A Target is also known as an Adapter or Slave.
- Producer – sends data onto the Ethernet/IP network
- Consumer – receives data from the Ethernet/IP network

Chapter 3. Ethernet/IP TLM Configuration

3.1. Configuration Steps

There are a number of steps to establish Ethernet/IP communications between SoftPLC and other devices:

- Module Installation - Ensure the driver and configuration file are installed in the SoftPLC
- Enable ETHER_IP TLM – Configure SoftPLC to load the Ethernet/IP driver at startup
- Configure ETHER_IP TLM – Edit the driver configuration file ETHER_IP.LST for your desired communications
- Configure SoftPLC Tags – Create the SoftPLC data table elements and identify them with the tagnames in the ETHER_IP.LST configuration file
- Create Ladder Logic (optional) – If desired, add ladder logic rungs to use the TLI's included in the TLM to monitor and/or control the ETHER_IP driver operation
- Configure other Devices – Following vendor instructions, set up the other networked devices to communicate with the SoftPLC as configured (not described in this document)

3.2. Module Installation

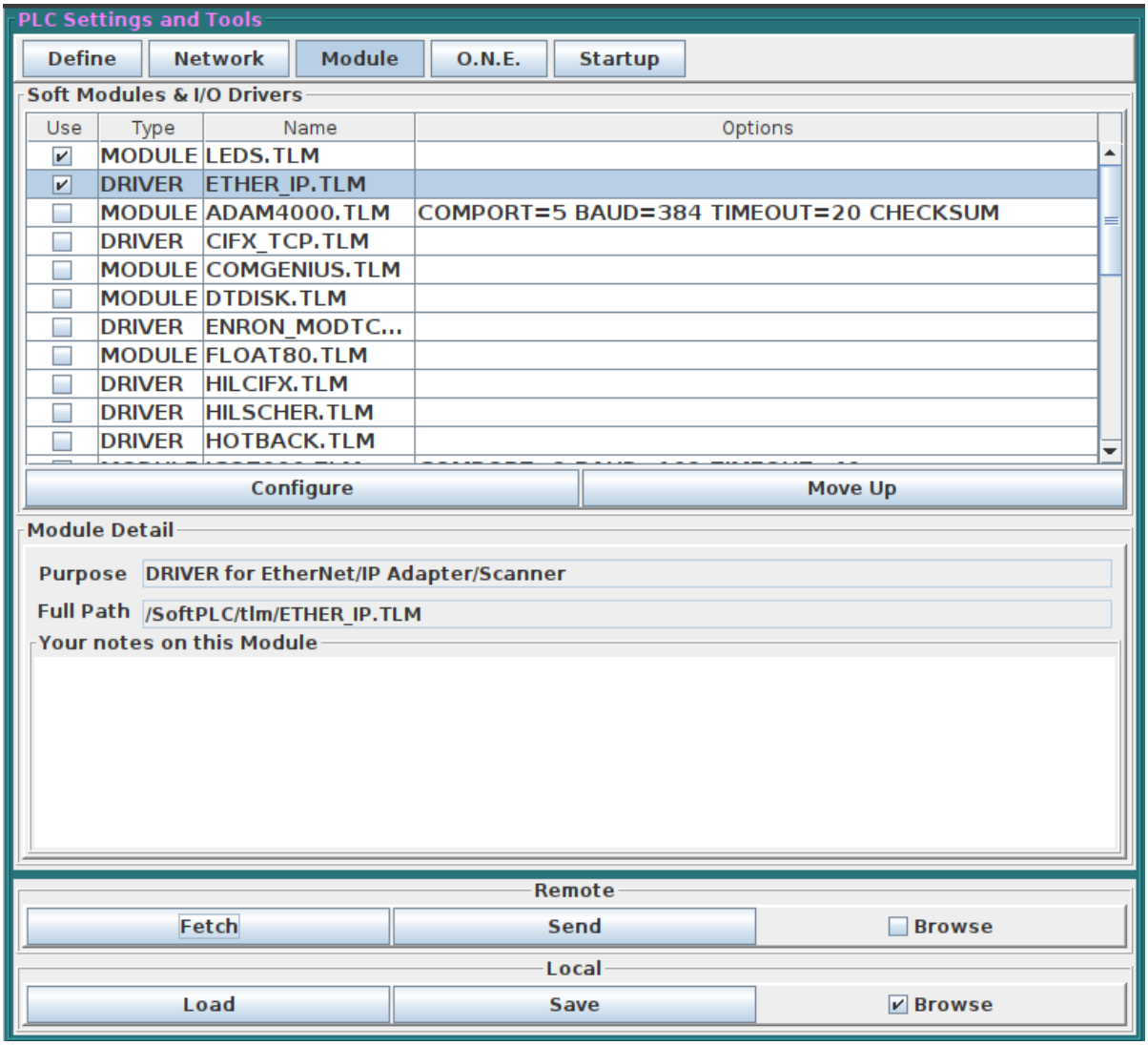
The TLM is a file named **ether_ip.tlm.so** and the configuration file is named **ETHER_IP.LST**. These files are pre-installed on the SoftPLC CPU or Gateway in the /SoftPLC/tlm directory.

In the event the files are not installed, you can use SFTP to transfer them from a PC to the SoftPLC.

3.3. Enable ETHER_IP TLM

The TOPDOC NexGen Manual and help system describe how to use the Module Editor to enable and configure TLM's. The following sections assume you understand TOPDOC NexGen's Module Editor, and other SoftPLC configuration procedures.

To enable the TLM within the SoftPLC, using TOPDOC NexGen, traverse to **PLC › Modules** and check **Use** for ETHER_IP.TLM. There are no **Options** for this TLM.



3.3.1. Save Enabled Modules

The **[Save]** button will write the MODULE.LST file to the development system's disk.
The **[Send]** button will write the MODULE.LST file to the runtime system's disk.

It is good practice to **both [Save]** and **[Send]** the edits, this way both your development system and the SoftPLC get a copy.

Whenever you **[Send]** a modified list of modules and/or their configuration files, you must restart or cycle power on the SoftPLC in order for the changes to take effect.

The next step is to **[Configure]** the TLM.

3.4. Configure ETHER_IP TLM

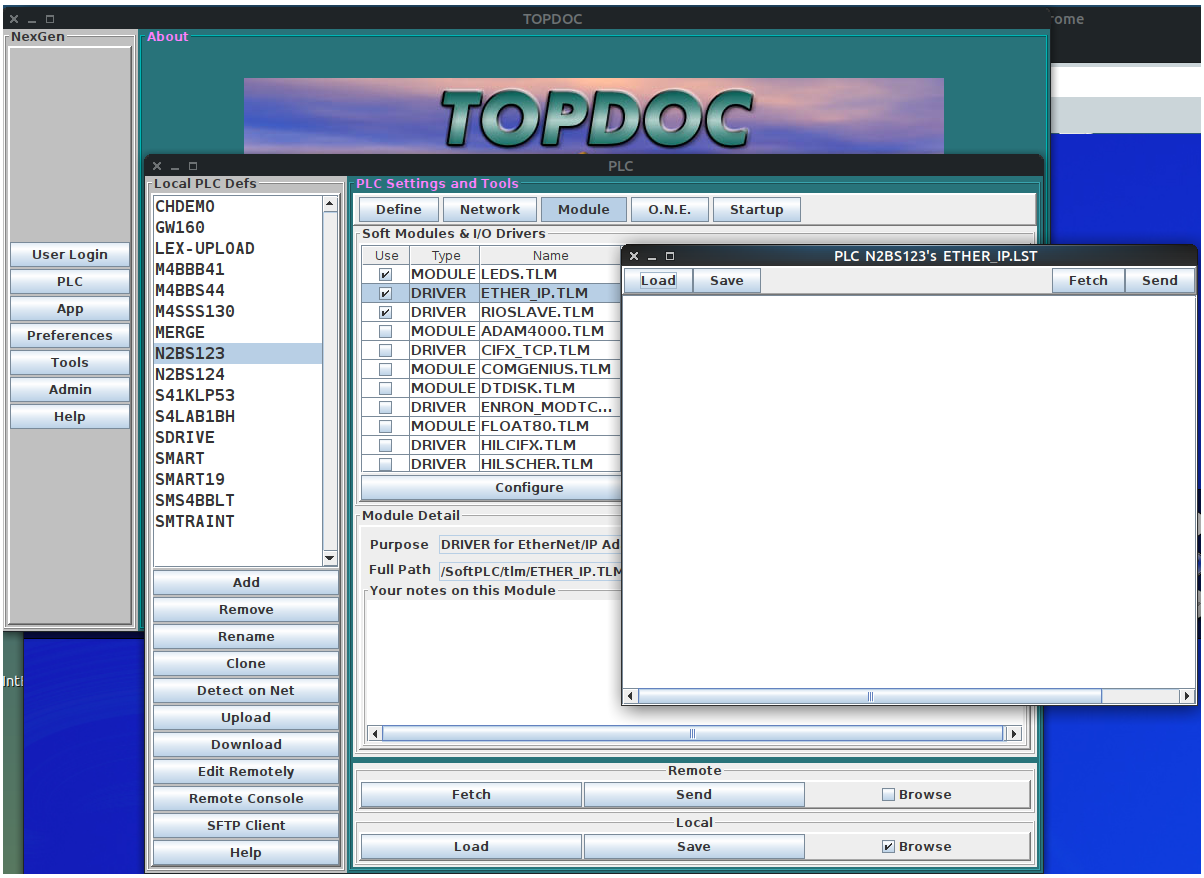
The configuration file for the ETHER_IP TLM is a text file called **ETHER_IP.LST**. The ETHER_IP.LST configuration file is used to set the debug level, specify which ethernet interface to use, define local assemblies, declare expectations for target (incoming) i/o connections, and define originator

(outbound) i/o connections. The last section of this manual includes the template ETHER_IP.LST file provided with each SoftPLC. The comments in the template file (lines preceded by a “#”) explain the file format, requirements, and options. The majority of the template file is comments, most of which are located at the start of the file. These comments act as a detailed help file for all the options supported by the TLM.

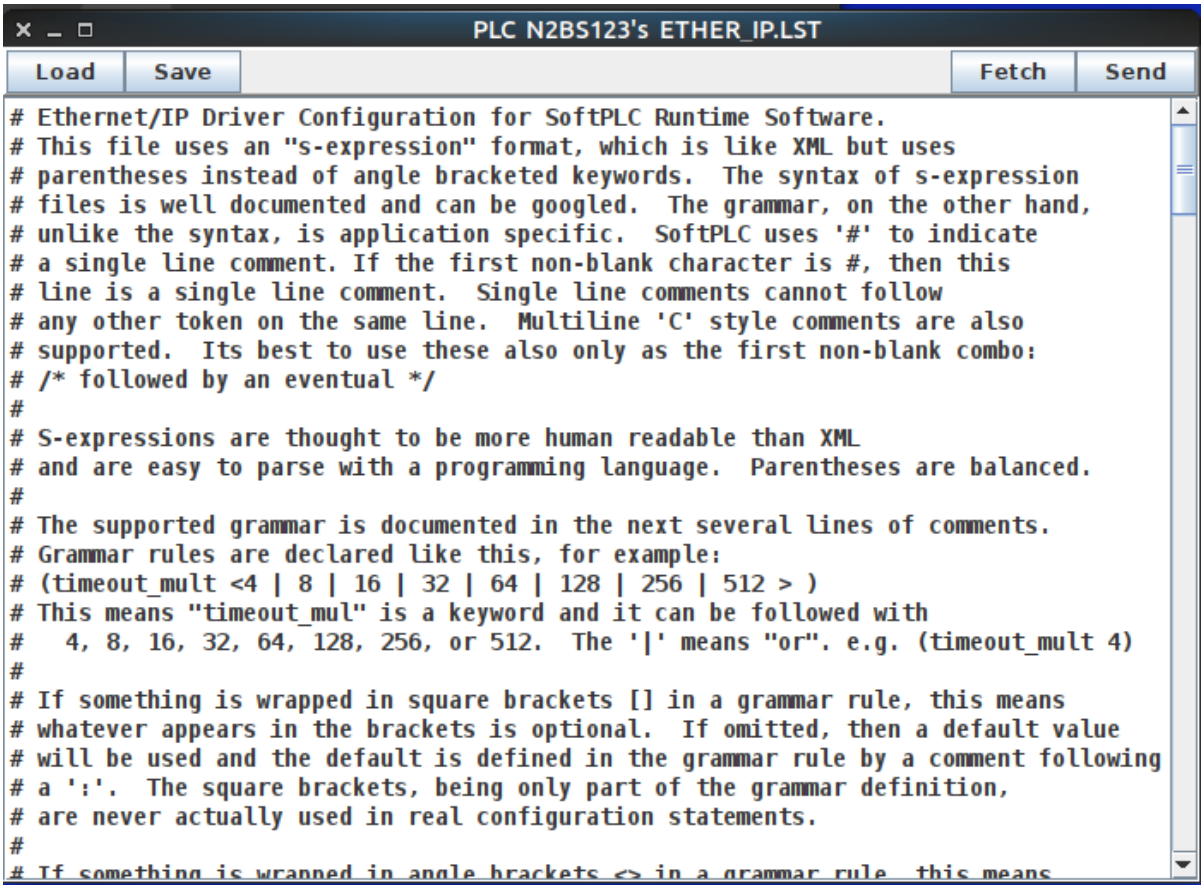
The active configuration information used by the TLM is located at the end of the template file.

3.4.1. ETHER_IP.LST Configuration Editor Usage

Click the **[Configure]** button below the list of modules to load the Configuration Editor for the ETHER_IP.LST file.



A template file is included in the SoftPLC. When connected to the SoftPLC, use the **[Fetch]** to load the template file into the Configuration Editor.

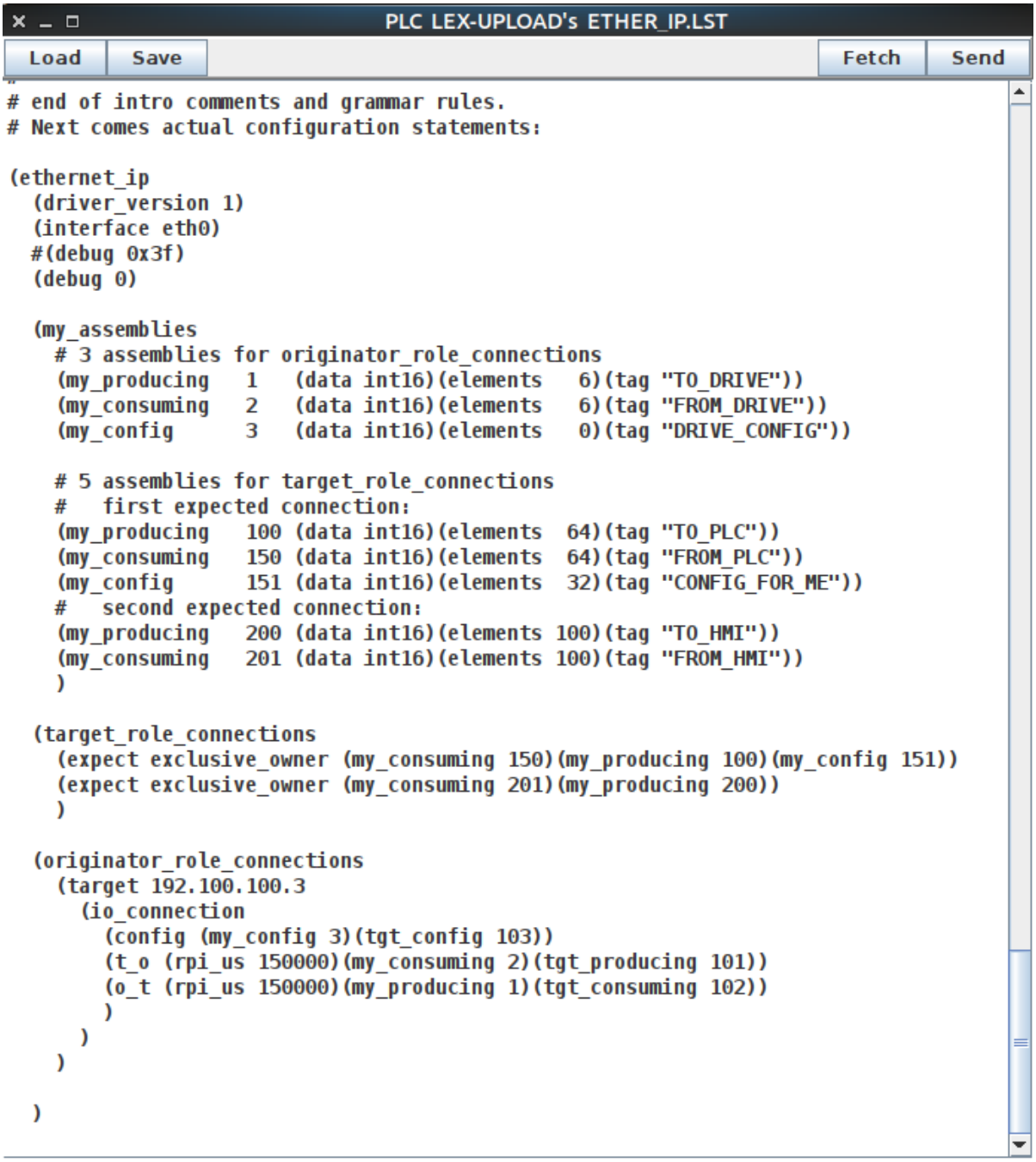


The best method for creating an ETHER_IP.LST file for your application is to start from the provided template file.

The **[Load]** button will load the ETHER_IP.LST file from the development system's disk.
The **[Save]** button will write the ETHER_IP.LST file to the development system's disk.
The **[Fetch]** button will load the ETHER_IP.LST file from the runtime system's disk.
The **[Send]** button will write the ETHER_IP.LST file to the runtime system's disk.

After you **[Send]** the configuration file, you must restart or cycle power on the SoftPLC in order for the changes to take effect.

After doing a **[Fetch]** or **[Load]** of the template ETHER_IP.LST file, use the left scroll bar to go to the end of the file so the "actual configuration statements" are displayed in the editor, as shown in the figure below.



3.4.2. ETHER_IP.LST Configuration File Structure

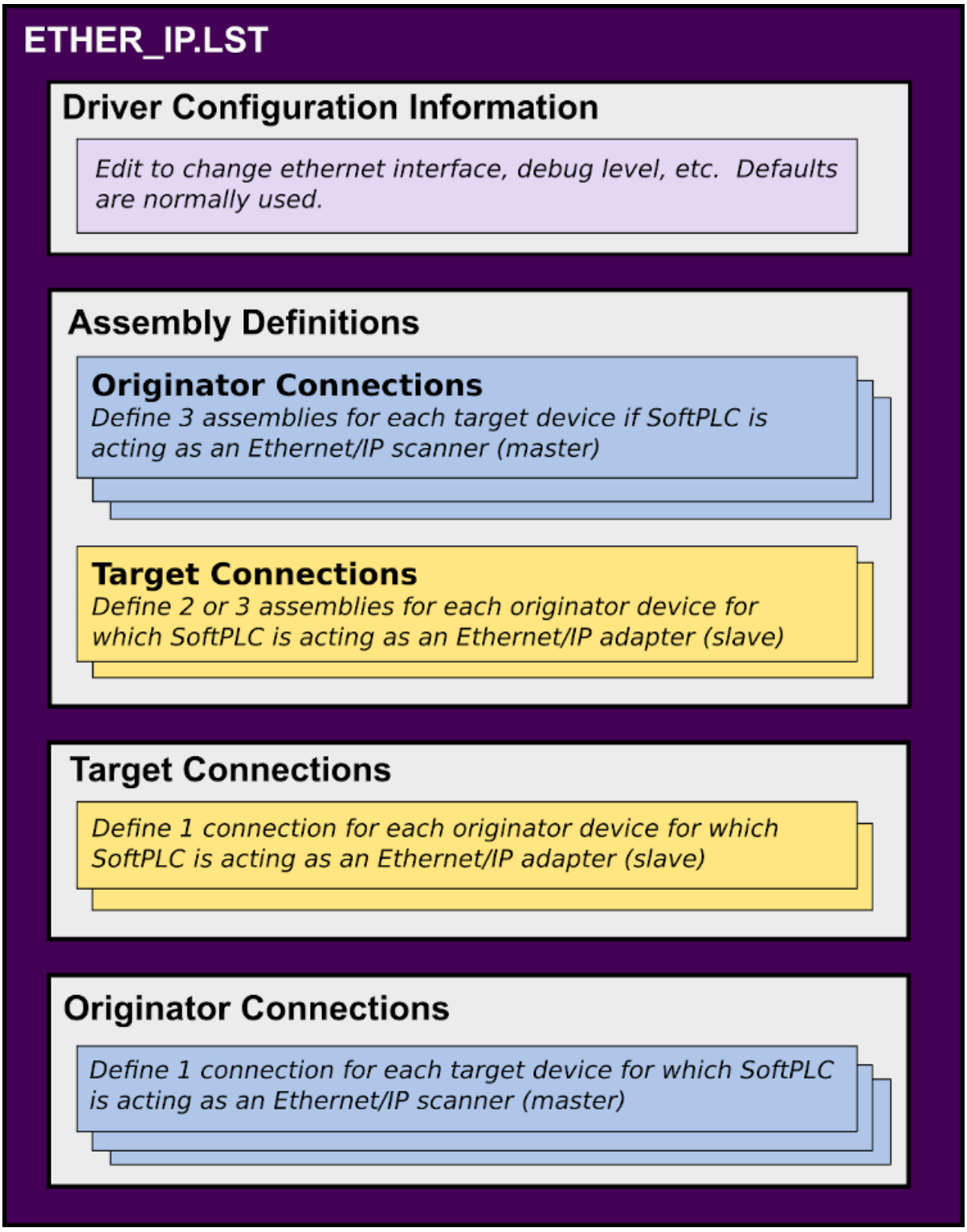
The figure below shows the structure of the ETHER_IP.LST configuration file. Depending on your application, you will need to either configure or de-activate certain sections of the template ETHER_IP.LST file.

The recommended method of de-activating sections in the template ETHER_IP.LST file is to add a “#” to the beginning of each line so that it becomes a comment.

- Driver Configuration Information is always required.
- If SoftPLC is acting as an Originator only, you will need to configure both the Originator Connections and the corresponding Assembly Definitions for each target device AND de-activate any Target Connections and the corresponding Assembly Definitions.
- If SoftPLC is acting as a Target only, you will need to configure both the Target Connections and

the corresponding Assembly Definitions for each originator device AND de-activate any Originator Connections and the corresponding Assembly Definitions.

- If SoftPLC is acting as both an Originator and a Target, you will need to configure all sections.



3.4.3. Driver Configuration Information

There are 4 elements in this section of ETHER_IP.LST. Some are optional. In most cases, you will not need to modify the default template information.

- driver version should be 1.

- interface for Smart or NeoPAC SoftPLC's should be ethernet0. If you are using a Hardbook SoftPLC, change the interface to eth0
- the debug 0 line (debugging mode off) should be enabled for normal operation.
    - In the event of communication problems, enabling the debug 0x3f instead provides additional communication logging to SoftPLC's system log.

If you enable debug mode by removing the # on **debug 0x3f**, make sure to also deactivate the debug 0 line by placing a # at the beginning of the line.

3.4.4. Originator Role(s) Configuration

If SoftPLC is acting as an Originator, there are 2 sections of ETHER_IP.LST that need to be configured **my_assemblies** and **originator_role_connections**. For each Target device, determine the IP address, and the type of data and number of elements to be sent from and received by the SoftPLC.

The template file has an example for SoftPLC acting as an Originator to one Target device. You can copy/paste the appropriate lines to support communication to multiple Targets.

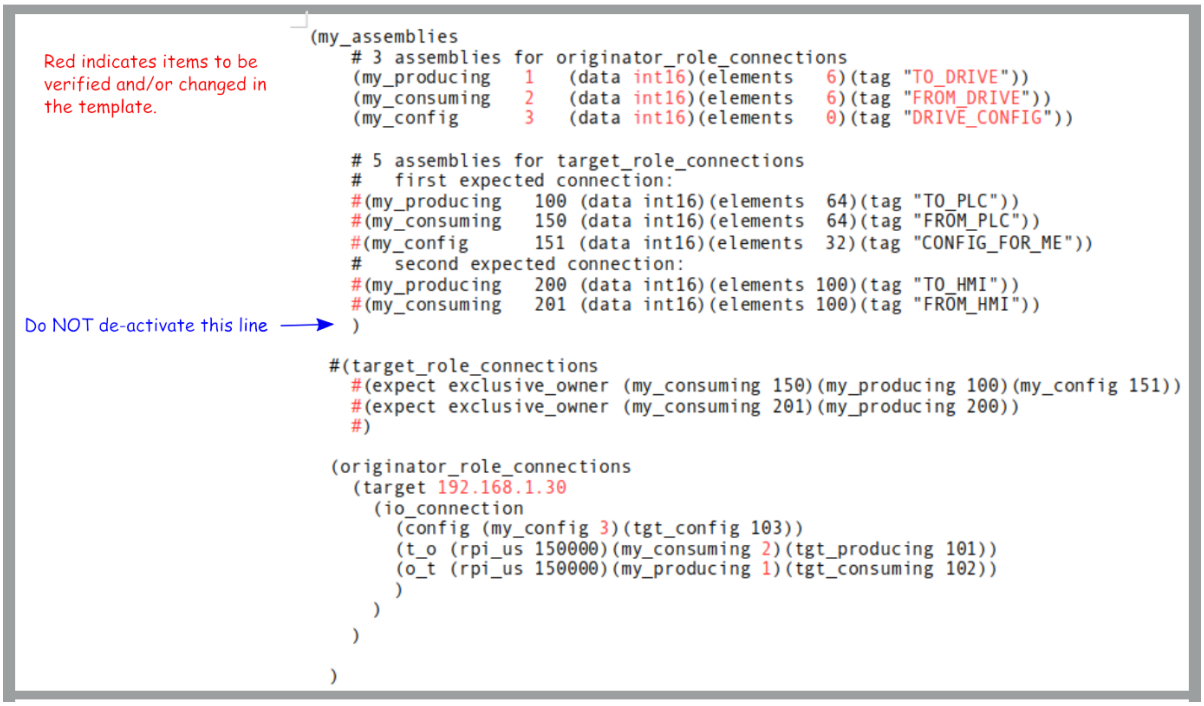
In the my_assemblies section, assign 3 assemblies for originator_role_connections for each target, and enter the type of data, and number of elements for each. You will also need to enter the desired SoftPLC tagnames for each assembly.

These tags and data table memory need to exist in the SoftPLC. This is described in the section titled "Tag Configuration".

In the originator_role_connections section, edit the target IP address(es), and enter the assembly numbers to match those entered in the my_assemblies section.

If this SoftPLC is an Originator only, deactivate the lines in the template file corresponding to target_role_connections.

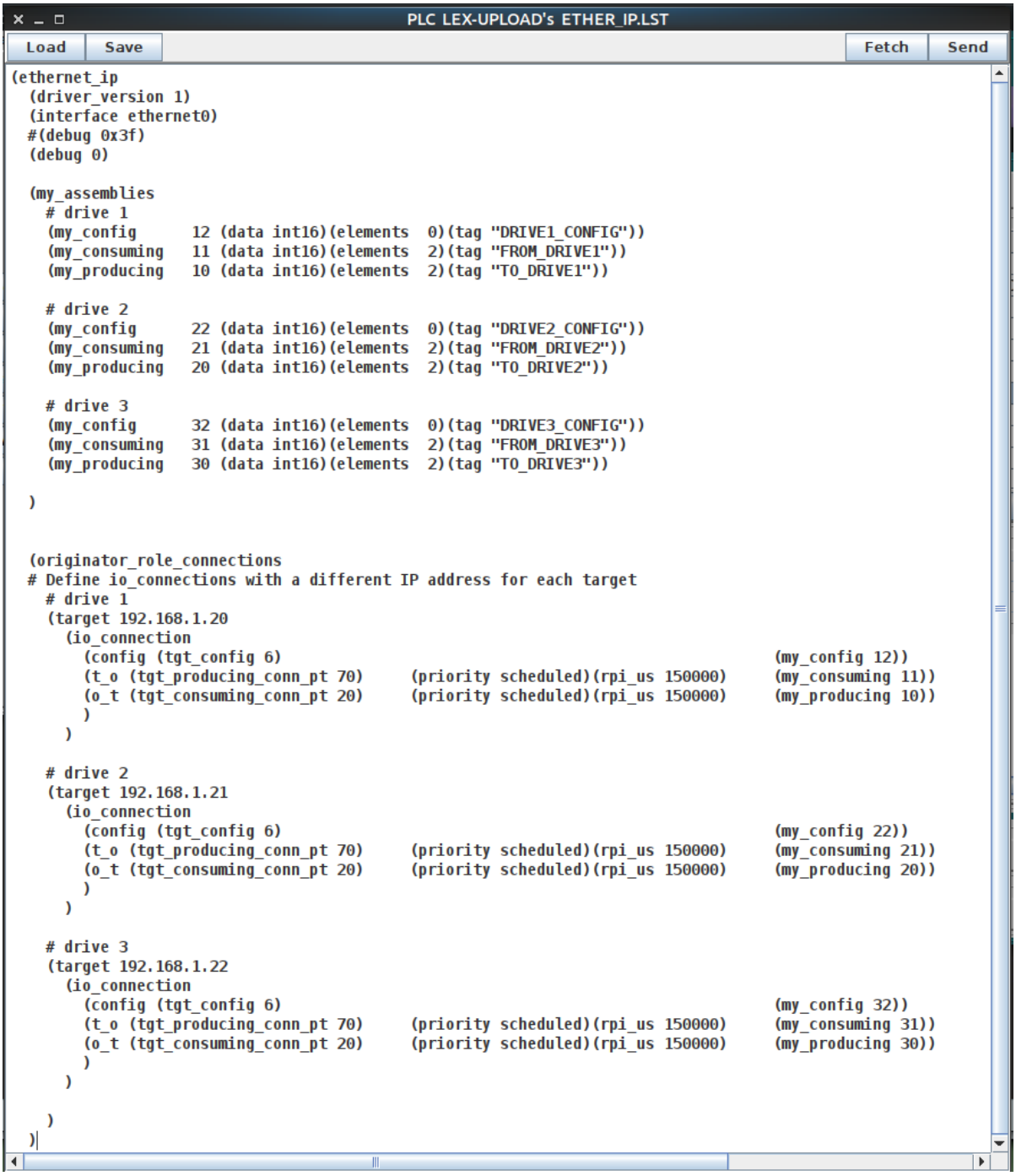
The image below depicts the Assembly and Connections Sections for an example ETHER_IP.LST for a SoftPLC that acts as an Originator to as ingle Target device, such as a Drive. Note that all the lines for target_role connections are commented out (de-activated), since this SoftPLC is an Originator only.



Example Configuration for Multiple Targets

The figure below shows an ETHER_IP.LST for a SoftPLC that is communicating as an Originator to 3 Ethernet/IP drive Targets.

This example is for Yaskawa Drives. Note that in the originator_role_connections section the type of assembly in this case is **tgt_producing_conn_pt**. This is because the Yaskawa drives require a 0x2c connection field path. This example also sets the priority to scheduled with an added parameter. The default template does not use this parameter, which sets the default priority of low.



3.4.5. Target Role(s) Configuration

If SoftPLC is acting as a Target, there are 2 sections of ETHER_IP.LST that need to be configured.

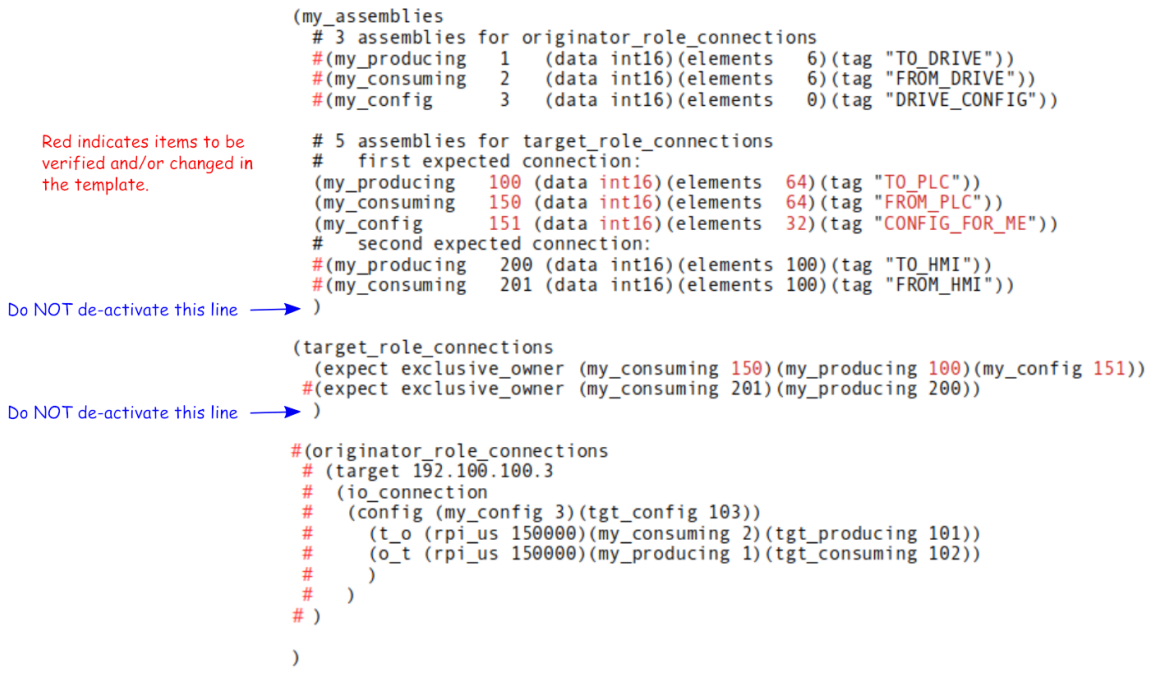
The template file has an example for SoftPLC acting as a Target to 2 Originator devices. You can deactivate the appropriate lines to support communication to a single Target.

For each Originator device, determine from its documentation whether it requires a Configuration assembly or not and the size of that assembly. Also determine the assembly numbers to be used for data to be sent and received. For your application, determine the type of data and number of elements to be sent to and received from the SoftPLC. In the my_assemblies section, assign the 2 (or 3) assemblies for target_role_connections for each Originator, and enter the type of data, and number of elements for each. You will also need to enter the desired SoftPLC tagnames for each

assembly. If this SoftPLC is a Target only, deactivate the lines in the template file corresponding to originator_role_connections.

These tags and data table memory need to exist in the SoftPLC. This is described in the section titled “Tag Configuration”.

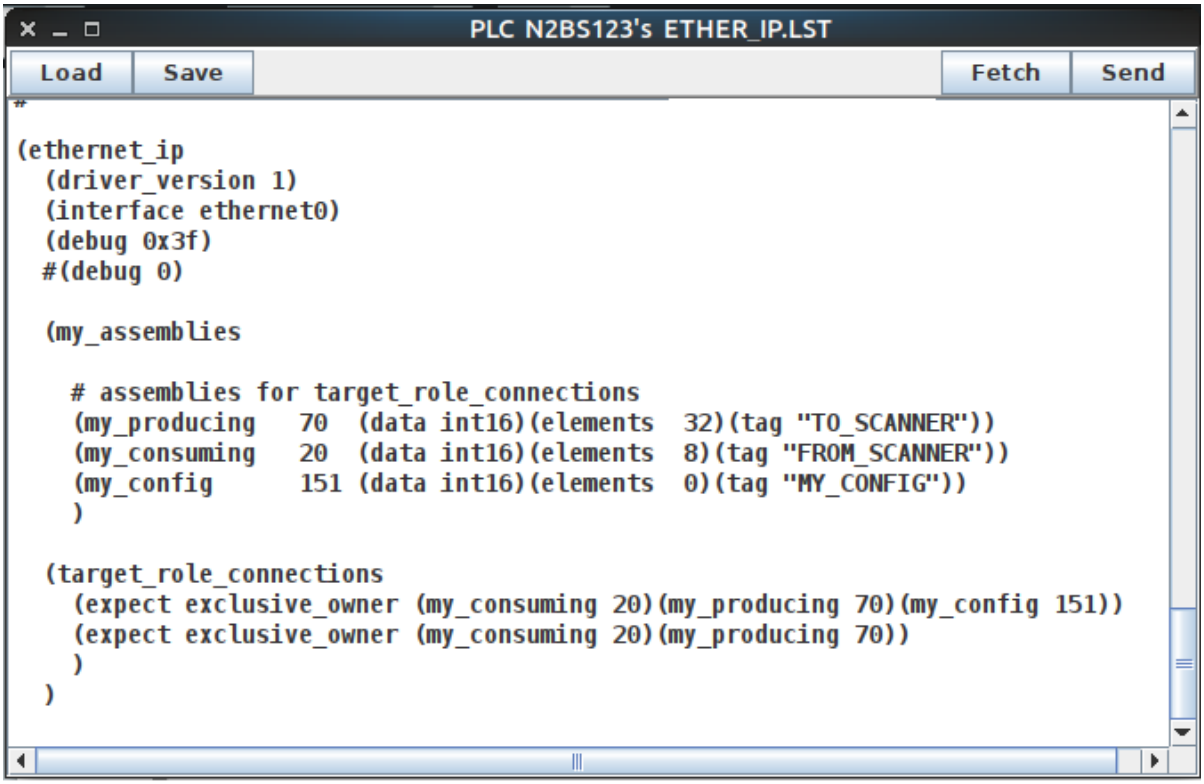
In the target_role_connections section, enter the assembly numbers to match those entered in the my_assemblies section. The image below depicts an example ETHER_IP.LST for a SoftPLC that acts as an Target only to 1 Orginator, such as a CompactLogix controller. Note that all the lines for originator_role connections are commented out (de-activated), since this SoftPLC is a Target only.

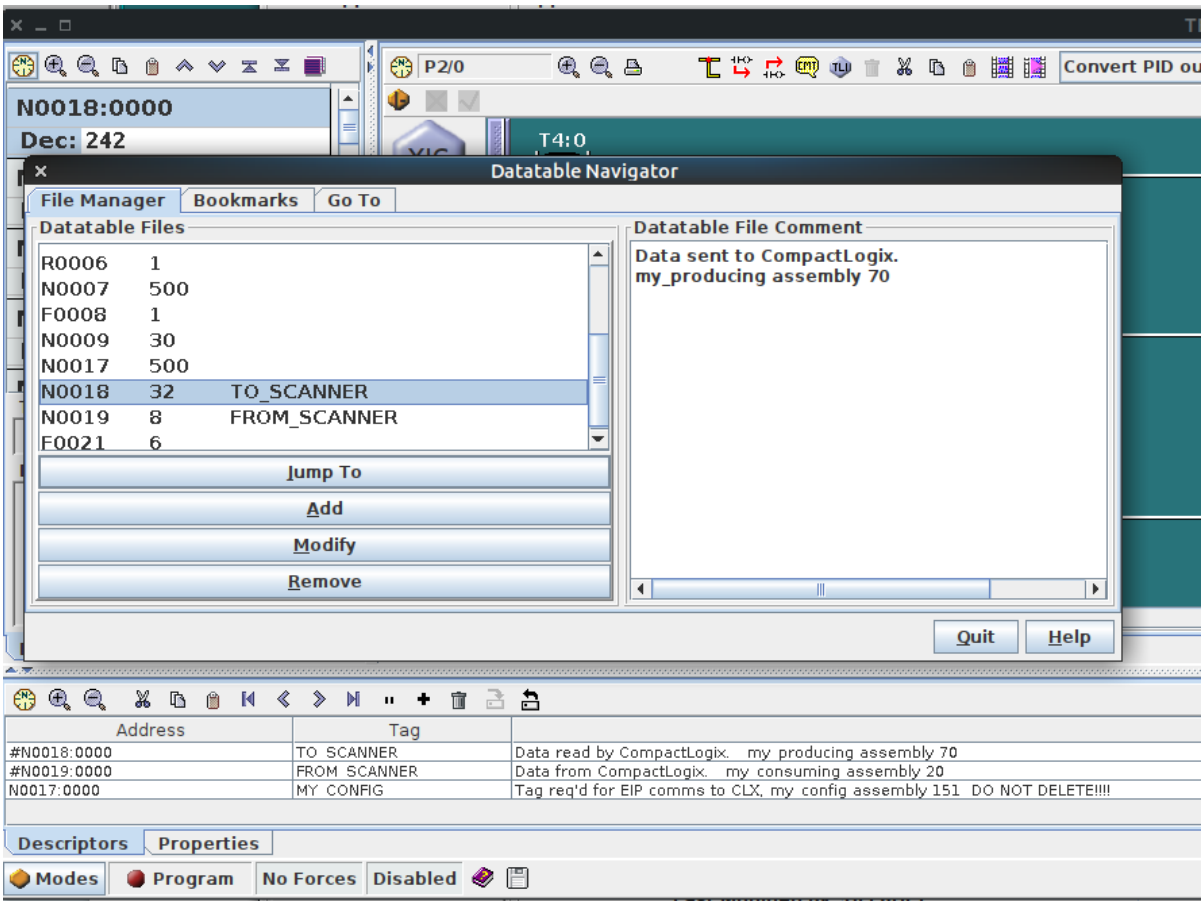


3.4.6. Configure SoftPLC Tags

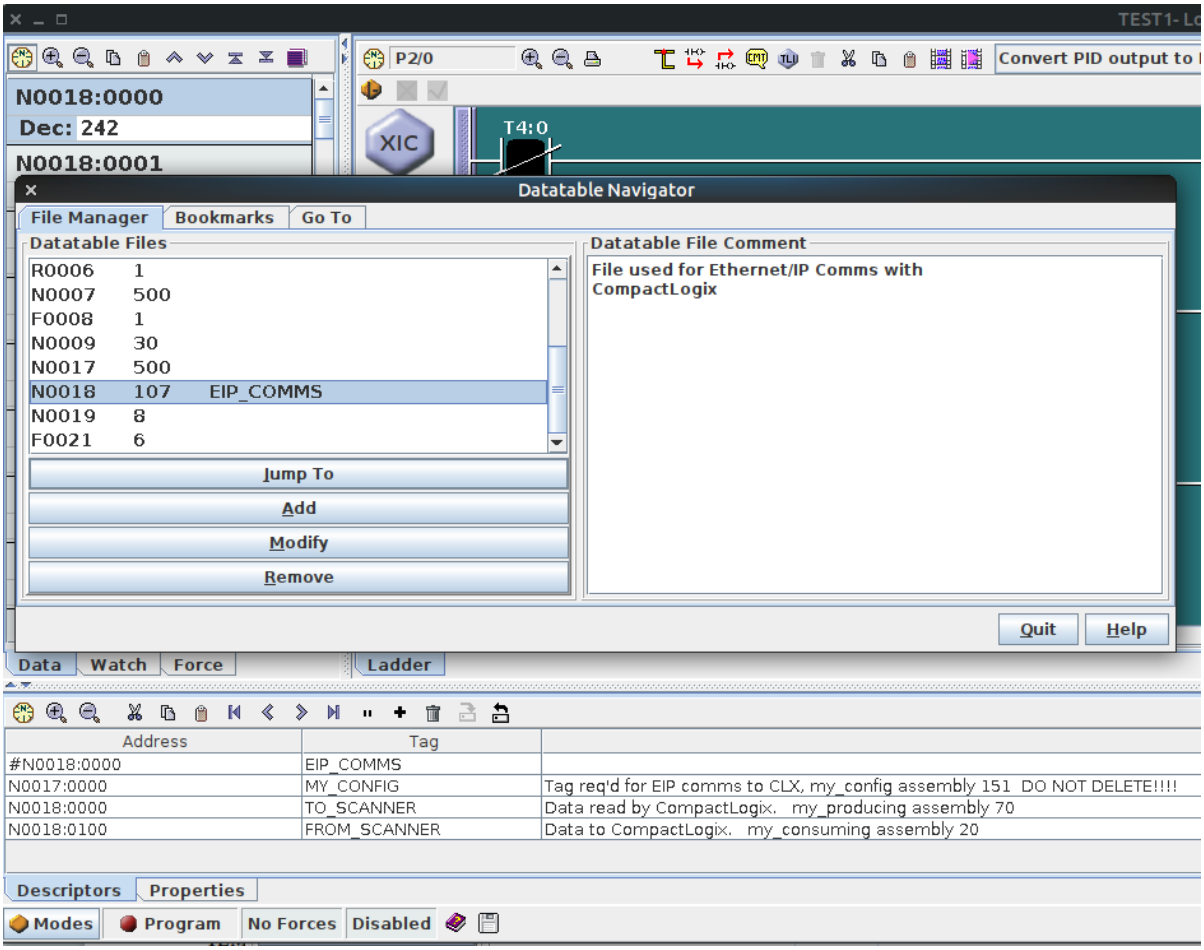
In the ETHER_IP.LST you identify tags that correspond to the assemblies you will be using for Ethernet/IP communication. Each of these tags will be tied to a data table address that is the start of an array of words/floats. These tags need to exist in the SoftPLC application that is selected to run at startup, and the necessary data table must exist to support each assembly.

Use TOPDOC NexGen to create the necessary data table files and the tagnames. The TOPDOC User Manual and help file describe how to use the data table and descriptor editors. The following 2 figures show an example of an ETHER_IP.LST and the corresponding tagnames and data table files required in the SoftPLC APP.





You can use the same data table file for multiple assemblies. Each tagname should correspond to the starting address of the block of words. Make sure each block of words is not used for other purposes. See example in the figure below.



If multiple TLM's are used, it is the user's responsibility to ensure that data table addresses are not assigned to more than one physical device. SoftPLC does not verify whether data table addresses have been defined to more than one driver.

Chapter 4. Monitor/Control TLM Operation

There are 3 TOPDOC Loadable Instructions (**TLI**)s that can be used in the SoftPLC's ladder logic to monitor and control the operation of the ETHER_IP TLM.

To use the TLI's in TOPDOC's offline APP editor, you need to have a copy of the ether_ip.tlm.so file in your /SoftPLC/tlm/ directory. This is installed by default with TOPDOC NexGen.

4.1. EIP_GETFAULTMAP

This TLI is a permissive (input) instruction that evaluates to either true or false depending on whether there are any (io_connections ...) which are in an unconnected state. That means that the output logic to the right of this permissive will be energized when there is some kind of communications failure. So it can act like an alarm.

Additionally, the datatable block given by the **Map** parameter will be filled with a bitmap containing one bit for each io_connection, and will turn on bits which represent the io_connections which are failed. Bit 0 of the first word at Map represents the first (io_connection ...) in the configuration file, bit 1 of the first word at **Map** represents the second (io_connection ...), etc. All io_connections across all targets are in this set. The length of the **Map** block is 128 bits fixed length. This is 128/16 = 8 words.

4.2. EIP_GETSTATS

This TLI is an output instruction that retrieves a block of 16 bit words into the **Result** parameter. Each word in the **Result** block is a count of how many times the TLM has lost contact with the respective io_connection.

Each instance of this instruction only involves the io_connections under one target, given by the **Target** parameter. In the **Target** STring you should put the same IP_ADDRESS or machine name that you used in the target's configuration in ETHER_IP.LST.

The io_connections defined within the configured target are implicitly numbered starting with index 0. The **IoConnIndex** is the start of a consecutive range of target io_connection indices, and will normally be 0 to indicate that you want the first io_connection under **Target**. **IoConnCount** is the number of io_connections that you want to fetch starting at **IoConnIndex**. It is the length of the range, and may not go beyond the number of io_connections given in the configuration file for **Target**, minus the starting **IoConnIndex**.

You will need a separate instance of this instruction for each target. This instruction can be used in conjunction with EIP_CLEARSTATS.

4.3. EIP_CLEARSTATS

This TLI is an output instruction that zeros one 16 bit counter word for each (io_connection ...) associated with a particular (target ...) given by the **Target** parameter. In the **Target** STring you should put the same IP_ADDRESS or machine name that you used in the target's configuration in ETHER_IP.LST.

The io_connections defined within the configured target are implicitly numbered starting with index 0.

You will need a separate instance of this instruction for each target. This instruction can be used in conjunction with EIP_GETSTATS.

Chapter 5. Debugging

This section gives tips on debugging problems on the Ethernet/IP network.

5.1. Isolating the Problem Target Node

During startup or when troubleshooting a problem node it is usually best to isolate the problem node. This means, look at it in isolation by making it the only active target on the network. You can keep the other targets connected, but use a temporary configuration file to announce to the TLM only the node that you are troubleshooting. All other nodes/targets will simply not be scanned.

Before you start debugging, you should use the configuration editor to Fetch and then Save your existing full blown configuration. Then, on your development system (Windows computer), temporarily copy the file \SoftPLC\plc\<PLCNAME>\ETHER_IP.LST to a safe place. Then, you can edit the configuration file temporarily and experiment freely. Later, restore by copying from the safe place back to \SoftPLC\plc\<PLCNAME>\ETHER_IP.LST. Then use the editor to Load then Send the file back down to SoftPLC. Remember that you have to restart SoftPLC after each configuration change.

5.2. Enable Debugging

The SoftPLC runtime engine constantly monitors its processes, and 'logs' these observations as process output. By default, these logs are minimal. However, for troubleshooting purposes, the logs can provide greater detail.

In the configuration file (ETHER_IP.LST), there is the attribute **debug**.

- A debug value of "0" represents the minimal detail to be logged.
- A debug value of "0x3f" represents the maximal detail to be logged.

5.3. View Debugging

Viewing these logs shall be completed at the command prompt of the SoftPLC system. To access the command prompt, log into the SoftPLC by either:

- (from Windows) use third-party 'PUTTY' application
- (from Linux) use SSH from Terminal application
- (TOPDOC 5.x) use Remote Console feature in the 'PLC' window

Default login credentials are as follows:
user: root
password: softplc

Once logged in, the logs can be viewed by executing one of the following:

(the '#' represents the prompt, and is not typed)

- For SoftPLC firmware 4.x
  - # logread
- For SoftPLC firmware 5.x
  - # journalctl -u softplc
    - You may need to use the arrow keys to scroll down to the end of the logs. The last logs are the most recent.

5.4. Direct Debugging to Text File

The previous sections have shown how to view the logs from the command prompt. However, recording the logs to text file format is, in the least, efficient for receiving support. Accomplishing this, much like viewing the logs, is firmware dependent (see following sections). Once the text file is created, it can be transferred via (S)FTP to the TOPDOC machine. A detailed explanation of (S)FTP transfers can be found in the TOPDOC User's Guide.

5.4.1. Direct Debugging output into a text file (SoftPLC 4.x)

1. Log into SoftPLC using either a) PUTTY from Windows or b) using ssh from Linux or c) at the command prompt of the SoftPLC system.
2. Run this command:
   # /etc/init.d/softplc.sh stop
3. Change into the /SoftPLC/run directory:
   # cd /SoftPLC/run
4. You can run SoftPLC from the command prompt now and redirect its output to an arbitrary file (named out.txt here). We put that file into the RAM disk which is anchored in the /tmp directory.
   # ./runsplc > /tmp/out.txt
5. Let this run for 5-60 seconds, then press control-C. Now you have the output captured in file /tmp/out.txt, each log entry will be captured in that file.
6. You can look at the file using the program named "less".
   # less /tmp/out.txt
   You can look at this output with the Ethernet/IP Specification, and the manual for your I/O module in hand. Press ESC when done.
7. You can make configuration file changes and Send them down to the SoftPLC. Then merely repeat the part of this process starting at step 4 above.
8. When done, remember to set debug back to "0", then you can start SoftPLC as a daemon either by a) power cycling the box or b) doing the following:
   # /etc/init.d/softplc.sh start

5.4.2. Direct Debugging output into a text file (SoftPLC 5.x)

1. Log into SoftPLC using either a) the remote console feature in TOPDOC's PLC window, b) PUTTY from Windows, or c) using ssh from Linux.

2. Run this command:
   # systemctl stop softplc
3. Change into the /SoftPLC/run directory:
   # cd /SoftPLC/run
4. You can run SoftPLC from the command prompt now and redirect its output to an arbitrary file (named out.txt here). We put that file into the RAM disk which is anchored in the /tmp directory.
   # ./runsplc > /tmp/out.txt
5. Let this run for 5-60 seconds, then press control-C. Now you have the output captured in file /tmp/out.txt, each request-response transaction will be captured in that file.
6. You can look at the file using the program named "less".
   # less /tmp/out.txt
   You can look at this output with the Ethernet/IP Specification, and the manual for your I/O module in hand. Press ESC when done.
7. You can make configuration file changes and Send them down to SoftPLC. Then merely repeat the part of this process starting at step 4 above.
8. When done, remember to set debug back to "0", then you can start SoftPLC as a daemon either by a) power cycling the box or b) doing the following:
   # systemctl start softplc

Chapter 6. Template ETHER_IP.LST File

Sample ETHER_IP.LST

```
# Ethernet/IP Driver Configuration for SoftPLC Runtime Software.
# This file uses an "s-expression" format, which is like XML but uses
# parentheses instead of angle bracketed keywords.  The syntax of s-expression
# files is well documented and can be googled.  The grammar, on the other hand,
# unlike the syntax, is application specific.  SoftPLC uses '#' to indicate
# a single line comment. If the first non-blank character is #, then this
# line is a single line comment.  Single line comments cannot follow
# any other token on the same line.  Multiline 'C' style comments are also
# supported.  Its best to use these also only as the first non-blank combo:
# /* followed by an eventual */
#
# S-expressions are thought to be more human readable than XML
# and are easy to parse with a programming language.  Parentheses are balanced.
#
# The supported grammar is documented in the next several lines of comments.
# Grammar rules are declared like this, for example:
# (timeout_mult <4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 > )
# This means "timeout_mul" is a keyword and it can be followed with
#   4, 8, 16, 32, 64, 128, 256, or 512.  The '|' means "or". e.g. (timeout_mult 4)
#
# If something is wrapped in square brackets [] in a grammar rule, this means
# whatever appears in the brackets is optional.  If omitted, then a default value
# will be used and the default is defined in the grammar rule by a comment following
# a ':'.  The square brackets, being only part of the grammar definition,
# are never actually used in real configuration statements.
#
# If something is wrapped in angle brackets <> in a grammar rule, this means
# it is a grouping.  Only one of the choices in a grouping may actually be used.
# Angle brackets, being only part of the grammar definition, are never actually
# used in real configuration statements.
#
# White spaces includeing tabs and newlines (CR, LF, etc.) are all ignored.
# Numbers may be provided either in decimal or in hex with a prefix of 0x.

#------ Grammar Rules: ----------
# (ethernet_ip                  ` is the master containing element and it may contain
#                               ` the indicated 6 top level elements.  Some of the
#                               ` 6 are optional, indicated with enclosing [].
#                               ` Every nested element is contained within this
#                               ` master element.
#   (driver_version ...)
#   (interface ...)
#   (debug ...)
#   [(plc2_compatibility_file ...)]
#   [(my_assemblies ...)]
#   [(target_role_connections ...)]
#   [(originator_role_connections ...)]
#   )  /* end of "ethernet_ip" s-expression element */
#
# Details on nested elements:
```

```
#
# (driver_version <VERSION_NUM>)            ` VERSION_NUM should be 1
# (interface <ETH_PORT>)                    ` which ethernet interface to use,
#                                           ` e.g. ethernet0 (on Smart4) or eth0 (on x86)
# (debug <BIT_SET>)                         ` turns on debugging features given in
#                                           ` the integer bitset.
# (plc2_compatibility_file 7)               ` the N file number which mimics a
#                                           ` PLC2 datatable for old PCCC commands.
#
# Section (my_assemblies ...) causes the creation of the listed assemblies in
# the CIP domain and ties them into the SoftPLC datatable based on what PLC
# address the "tag" is defined to in the descriptor table.  Assemblies each
# must have a unique number from 1-65000, called an assembly instance id.
# These ids are then used in the two sections named
# target_role_connections and originator_role_connections which follow.
#
# (my_assemblies                            ` gives the CIP assemblies that the
#                                           ` TLM should create
#   <(my_producing ...) |
#    (my_consuming ...) |
#    (my_cfg ...) |
#    (my_explict ...)> ...
#   ) /* end of my_assemblies */
#
#   (my_producing <ASSEMBLY_ID>             ` flows from the datatable onto the network
#     (data <int16 | float32>)(elements <ELEMENT_COUNT>)(tag <TAG_NAME>)
#     )
#   (my_consuming <ASSEMBLY_ID>             ` flows from network into datatable
#     (data <int16 | float32>)(elements <ELEMENT_COUNT>)(tag <TAG_NAME>)
#     )
#   (my_config <ASSEMBLY_ID>                ` is bidirectional with datatable
#     (data <int16 | float32>)(elements <ELEMENT_COUNT>)(tag <TAG_NAME>)
#     )
#   (my_explicit <ASSEMBLY_ID>              ` is bidirectional with datatable
#     (data <int16 | float32>)(elements <ELEMENT_COUNT>)(tag <TAG_NAME>)
#     )
#
# Section (target_role_connections ...) is a list of i/o connections that should
# be accepted should we see an inbound forward_open containing a matching
# combination of assembly ids.
# (my_consuming ...) and (my_producing ...) and (my_config ...) are from
# my perspective within this network node.
#
# (target_role_connections
#   (expect exclusive_owner                 ` which kind of i/o connection, only
#                                           ` exclusive_owner for now
#     (my_consuming 150)                    ` which consuming assembly id
#     (my_producing 100)                    ` which producing assembly id
#     [(my_config 151)]                     ` optional config assembly id
#     )
#   (expect exclusive_owner (my_consuming 150)(my_producing 100))
#   )   /* end of target_role_connections */
#
#
# Section (originator_role_connections ...) lists client connections which
```

```
# will be made and manageed as a scanner.
# (originator_role_connections
#   (target ...)
#   [(target ...)] ...
#   )   /* end of originator_role_connections */
#
# 'target' is a TCP node that is capable of being an Ethernet/IP explicit message
#  server that supports the forward_open command.  One or more forward_open
#  commands will be sent to this node in order to setup the requested number
#  of io_connections, which are all UDP based.
#
# (target <IP_ADDRESS | HOSTNAME>           ` HOSTNAME ok if findable via DNS,
#                                           ` else provide an IP_ADDRESS
#   [(tcp_connect_timeout_ms <MSECS>)]      ` TCP connect() time limit of
#                                           ` originating : 5000 is default
#   [(tcp_connect_retry_delay_ms <MSECS>)]  ` How long after failed TCP connection
#                                           ` to try again. : 5000 is default
#   (io_connection ...)
#   [(io_connection ...)] ...
#   )   /* end of target */
#
# (io_connection
#   [(conn_path ...)]                       ` Hex bytes in CIP segment form to
#                                           ` precede the 1-3 Application Paths.
#                                           ` If used at all, probably just a port path.
#   [(fwd_open_timeout_ms <MSECS>)]         ` how long client waits for server after
#                                           ` sending forward_open : 1000 is default
#   [(io_timeout_mult <4|8|16|32|64|128|256|512> )] `see CIP docs for forward_open.
#                                           ` Actual io_connection timeout is unique for
#                                           ` each connection half, and is rpi_usecs times
#                                           ` this number : 4 is default
#   [<cyclic | change_of_state>]            ` CIP connection trigger
#                                           ` type : cyclic is default
#   [<class0 | class1>]                     ` CIP transport class : class1 is default
#   [(config ...)]                          ` configuration one shot transfer
#   [(t_o ...)]                             ` target to originator connection half,
#                                           ` if missing means no t->o transfer
#   [(o_t ...)]                             ` originator to target connection half,
#                                           ` if missing means no o->t transfer
#   )   /* end of io_connection */
#
# (config
#   (my_config <ASSEMBLY_ID>)               ` A block of data to send to the target,
#                                           ` or it can be zero length also.
#   [(tgt_config <ASSEMBLY_ID>)]            ` Which assembly is to receive it.
#                                           ` May be omitted if this info is in (conn_path ...)
#   [(size_bytes <NUM_BYTES>)]              ` If missing then the size of
#                                           ` my_config will be used.
#   )
#
# (t_o                                      ` target to originator connection half
#   (rpi_us <NUM_USECS>)                    ` Requested Packet Interval in usecs not msecs
#   [<p_to_p | multicast>]                  ` peer to peer or multicast
#                                           ` : p_to_p is default
#   [(priority <low | high | scheduled | urgent>)]
```

```
#                                       ` : low is default
#   [(rt_fmt <32_bit_header | modeless>)]   ` CIP Vol1 3-6.1
#                                       ` : "modeless" is default for t_o
#   (my_consuming <ASSEMBLY_ID>)            ` my assembly which is the consumer
#   [(tgt_producing <ASSEMBLY_ID>) | (tgt_producing_conn_pt <ASSEMBLY_ID>)]
#                                       ` the producing assembly number or the
#                                       ` producing connection point within the target.
#                                       ` Some target adapters require 0x2C as connection
#                                       ` path field prefixes rather than 0x24.
#                                       ` tgt_producing generates 0x24.
#                                       ` tgt_producing_conn_pt generates 0x2C.
#   [(size_bytes <NUM_BYTES>)]              ` If zero then this is a heartbeat half.
#                                       ` If missing then the size of
#                                       ` my_consuming will be used.
#   )   /* end of t_o */
#
# (o_t                                  ` originator to target connection half
#   (rpi_us <NUM_USECS>)                  ` Requested Packet Interval in usecs not msecs
#   [<p_to_p | multicast>]                ` peer to peer or multicast
#                                       ` : p_to_p is default
#   [(priority <low | high | scheduled | urgent>)]
#                                       ` : low is default
#   [(rt_fmt <32_bit_header | modeless>)]   ` CIP Vol1 3-6.1
#                                       ` : "32_bit_header" is default for o_t
#   (my_producing <ASSEMBLY_ID>)            ` my assembly which is the producer
#   [(tgt_consuming <ASSEMBLY_ID>) | (tgt_consuming_conn_pt <ASSEMBLY_ID>)]
#                                       ` the consuming assembly number or the
#                                       ` consuming connection point within the target.
#                                       ` Some target adapters require 0x2C as connection
#                                       ` path field prefixes rather than 0x24.
#                                       ` tgt_consuming generates 0x24.
#                                       ` tgt_consuming_conn_pt generates 0x2C.
#   [(size_bytes <NUM_BYTES>)]              ` If zero then this is a heartbeat half.
#                                       ` If missing then the size of
#                                       ` my_producing will be used.
#   )   /* end of o_t */
#
# end of intro comments and grammar rules.
# Next comes actual configuration statements:

(ethernet_ip
  (driver_version 1)
  (interface ethernet0)
  #(debug 0x3f)
  (debug 0)
  #(plc2_compatibility_file 7)

  (my_assemblies
    # 3 assemblies for originator_role_connections
    (my_producing   1   (data int16)(elements   6)(tag "TO_DRIVE"))
    (my_consuming   2   (data int16)(elements   6)(tag "FROM_DRIVE"))
    (my_config      3   (data int16)(elements   0)(tag "DRIVE_CONFIG"))

    # 5 assemblies for target_role_connections
    #   first expected connection:
```

```
  (my_producing   100 (data int16)(elements  64)(tag "TO_PLC"))
  (my_consuming   150 (data int16)(elements  64)(tag "FROM_PLC"))
  (my_config      151 (data int16)(elements  32)(tag "CONFIG_FOR_ME"))
  #   second expected connection:
  (my_producing   200 (data int16)(elements 100)(tag "TO_HMI"))
  (my_consuming   201 (data int16)(elements 100)(tag "FROM_HMI"))
  )

(target_role_connections
  (expect exclusive_owner (my_consuming 150)(my_producing 100)(my_config 151))
  (expect exclusive_owner (my_consuming 201)(my_producing 200))
  )

(originator_role_connections
  (target 192.168.1.99
    # The (tgt_*) elements may be used to create fragments of the connection path.
    # Alternatively, you may omit the (tgt_*) elements and instead supply a
    # (conn_path) element for the whole connection path.
    (io_connection
      (config (my_config 3)(tgt_config 103))
      (t_o (rpi_us 150000)(my_consuming 2)(tgt_producing 101))
      (o_t (rpi_us 150000)(my_producing 1)(tgt_consuming 102))
      )
    )
  )

)
```

Appendix A: SoftPLC as Target to a Logix PLC

SoftPLC can be set up as a Target to a Rockwell Logix controller as a **Generic I/O Device**.

For Ethernet/IP, an "INPUT" assembly is an input to the scanner. An OUTPUT assembly is an output from the scanner. For Ethernet/IP, the perspective is always taken from that of the scanner, not the adapter. SoftPLC is an adapter in this case.

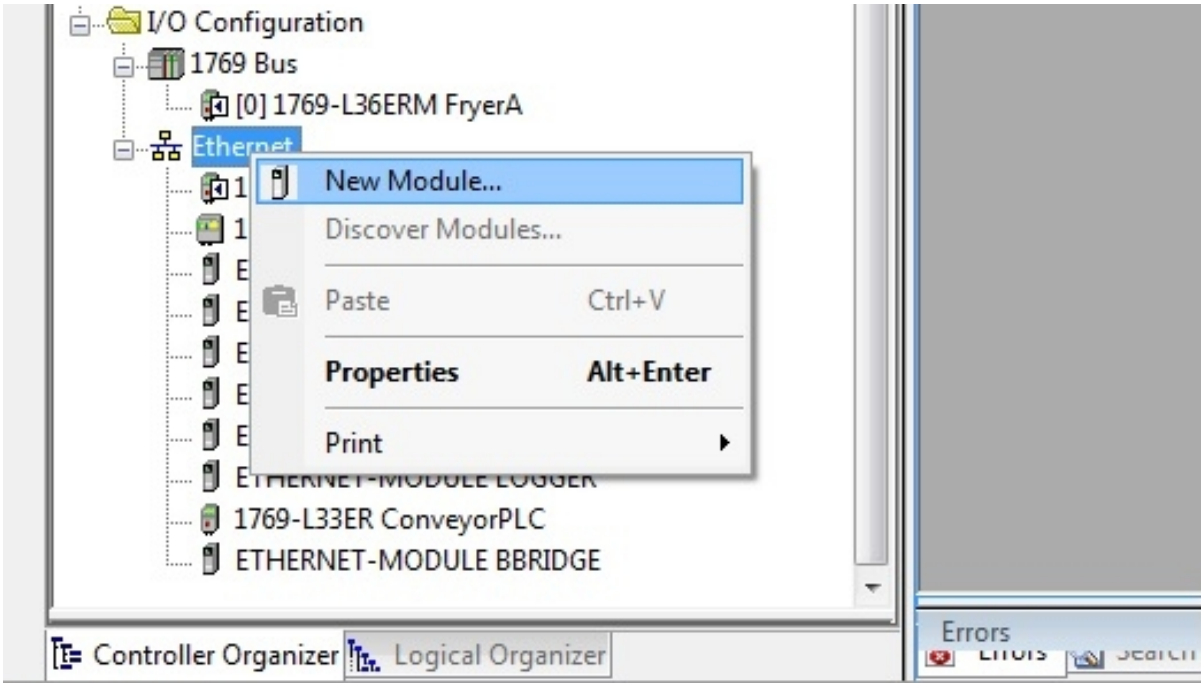
A.1. Configuring Logix to Communicate with SoftPLC as a Generic Ethernet Module

This section uses an example for an application called “LOGGER” to illustrate the required setup in RSLogix for this type communication.

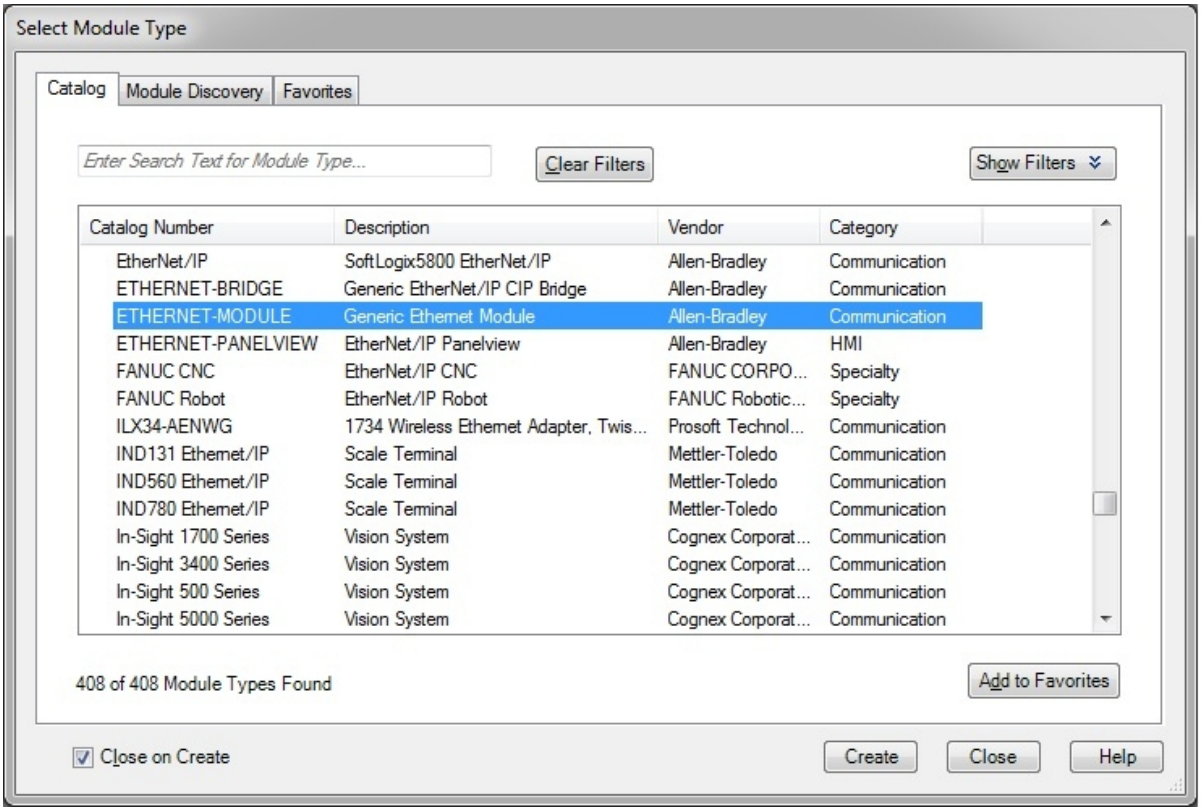
Assembly Instance Data for LOGGER Example:

- COM format → Integer (16 bit)
- Input Assembly → 100, Size 64 (data from SoftPLC, my_producing assembly in ETHER_IP.LST)
- Output Assembly → 150, Size 64 (data to SoftPLC, my_consuming assembly in ETHER_IP.LST)
- Configuration Assembly → 151, Size 0
- Connection Path RPI → 250ms

Step 1 – Create a new Module:

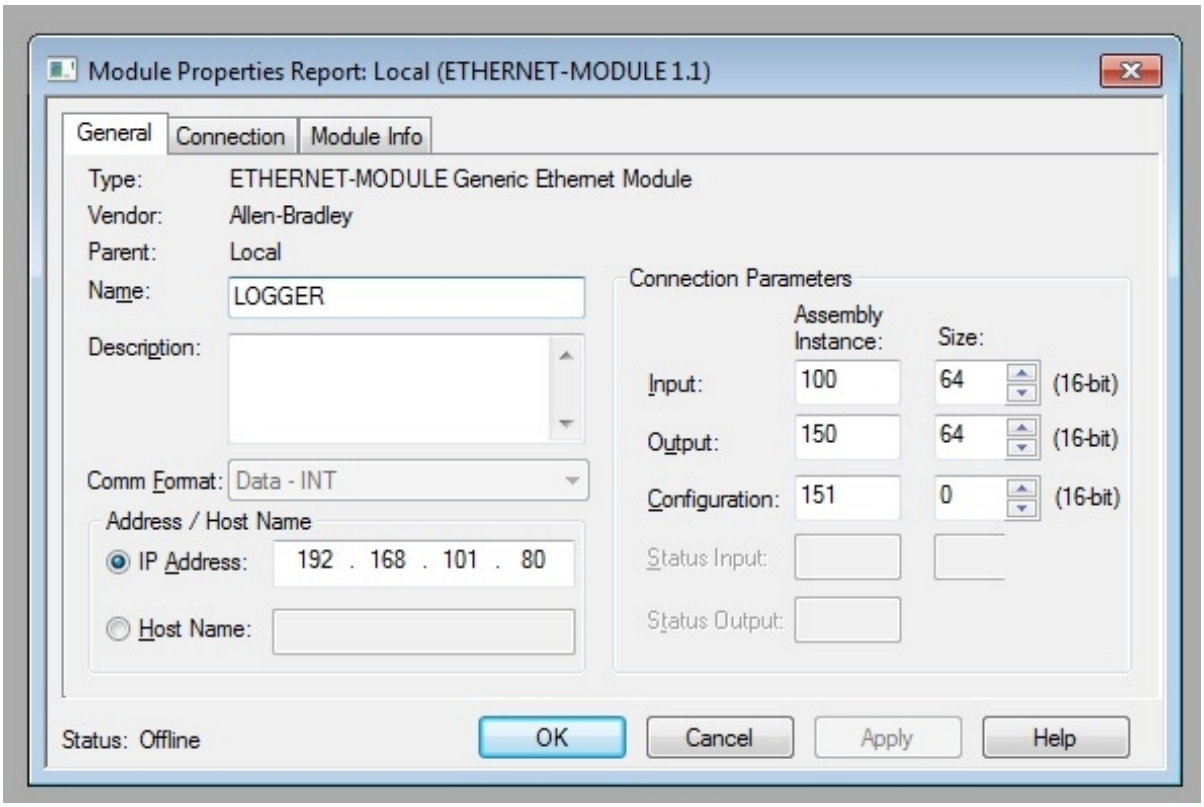


Step 2 – Select **Generic Ethernet Module** from the new module configuration window:

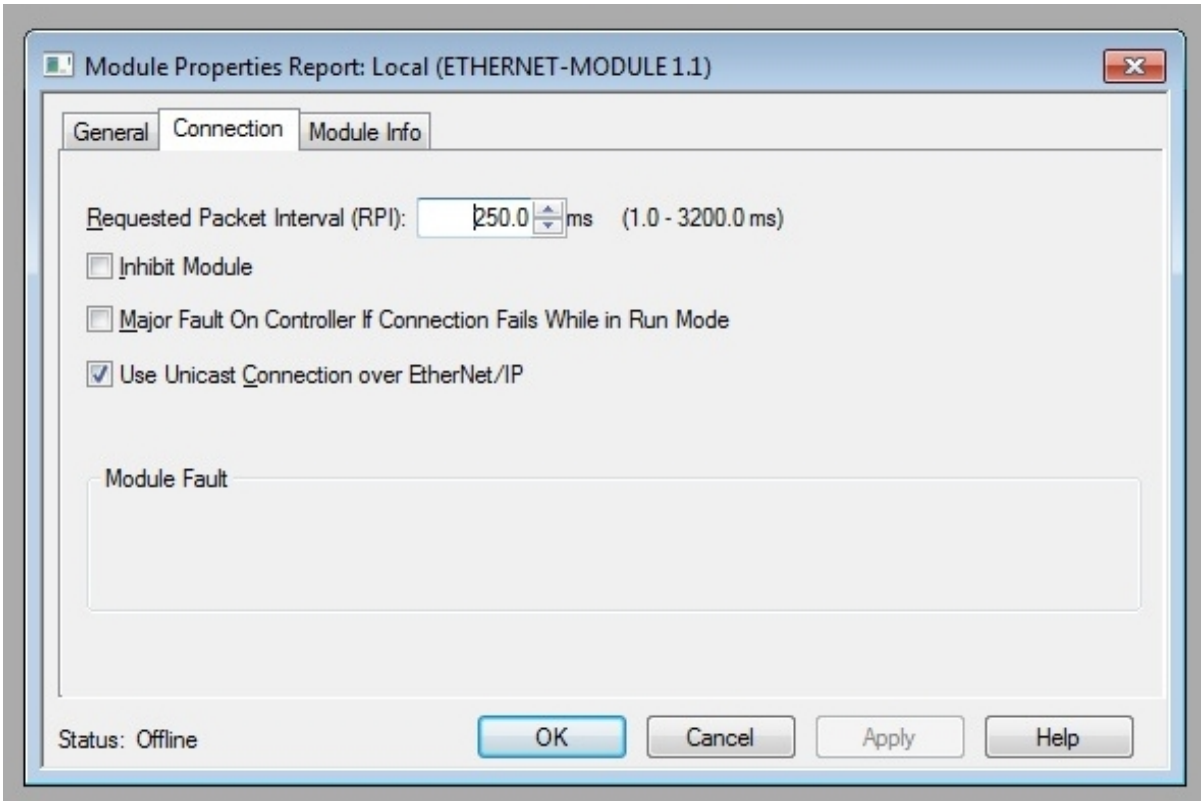


Step 3 - Enter the Module's General Properties:

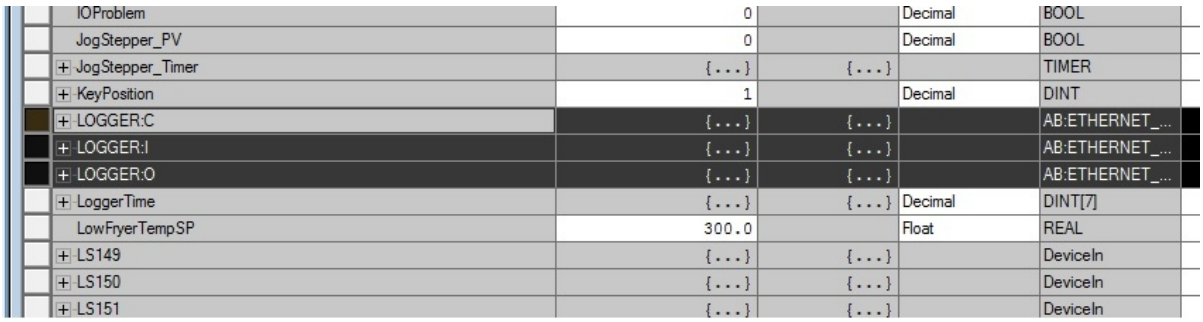
Make sure to select 16 bit INT for the Comm Format and for the format of each of the Connection Parameters, including the Configuration even though it is a size of 0.



Step 4 - Enter the Module Connection Properties:



Step 5 - Controller Tags are created:



A.2. Configuring Logix to Communicate with SoftPLC Using Messaging

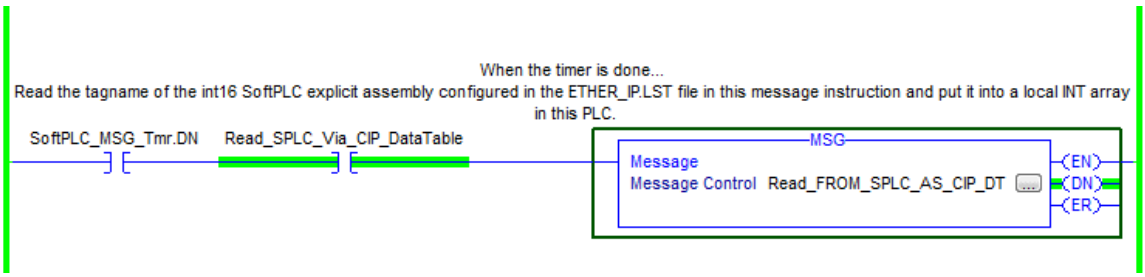
As an Ethernet/IP Target, SoftPLC can normally be configured as a “Generic Ethernet Module” when communicating with an Ethernet/IP Scanner, as described in the previous section.

However, some firmware revisions of Rockwell Logix products* do not conform to the ODVA Ethernet/IP specification with regard to communicating with “Generic Ethernet Modules”. This section provides information on how to set up the SoftPLC and Logix PLC using Message Instructions as an alternative method of communicating, in cases where the “Generic Ethernet Module” method does not work.

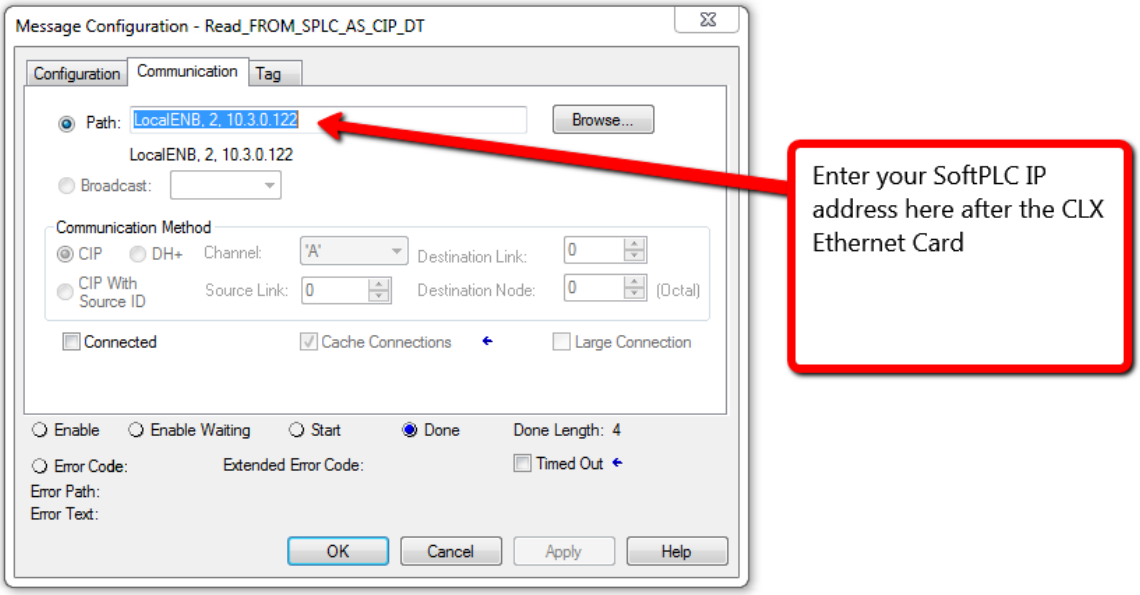
- Known firmware revisions that do not conform to the ODVA Ethernet/IP specification with regard to communicating with “Generic Ethernet Modules” are CompactLogix PAC’s with firmware 2x.xx or earlier. Some versions of CompactLogix with firmware revisions 3x.xx may also have this deficiency.

A.2.1. Read From SoftPLC via CIP Data Table Read

Enter the ladder logic shown in the figure below. This will initiate a READ of the configured target_role explicit assembly from SoftPLC into the Logix PLC.

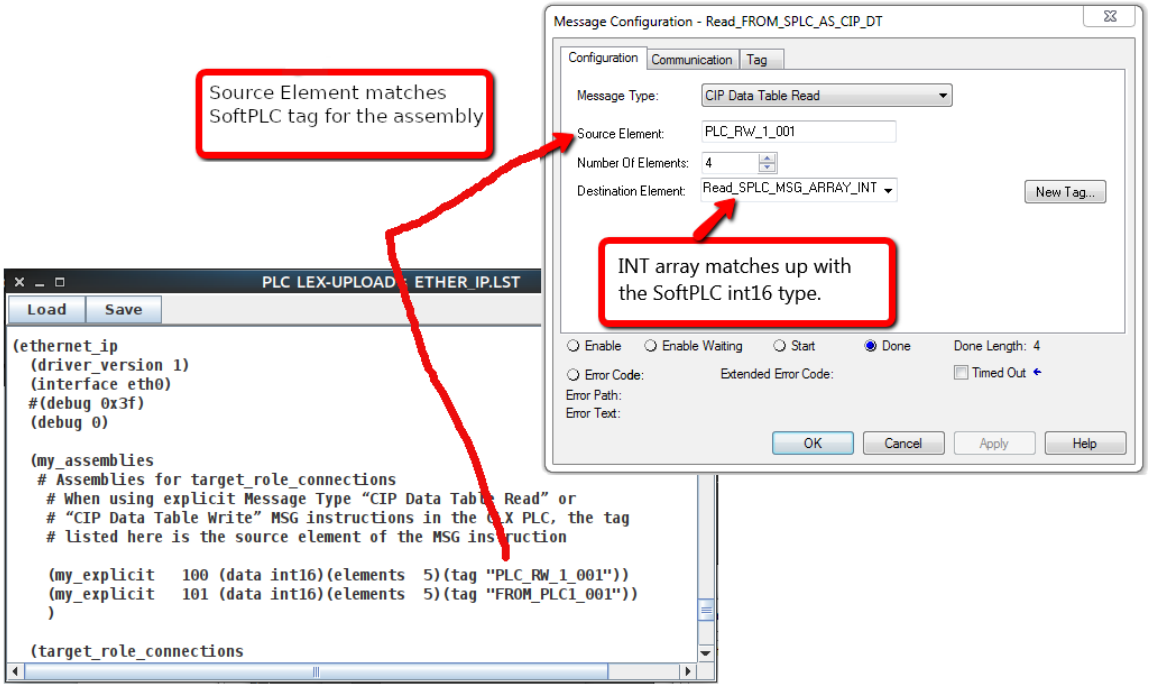


Configure the Message Instruction. The Path is the SoftPLC IP address after the CLX Ethernet interface ID.



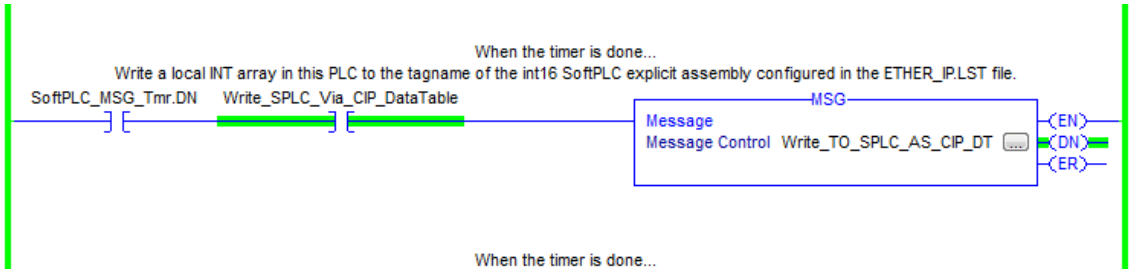
In RSLogix, the Source Element tag needs to match the tagname you will use in ETHER_IP.LST, and the Destination Element INT array needs to be one the matches the SoftPLC int16 type.

In the SoftPLC ETHER_IP.LST, configure a target assembly that supports the MSG instruction communications. This is an “explicit” type assembly, which is a bi-directional type assembly.

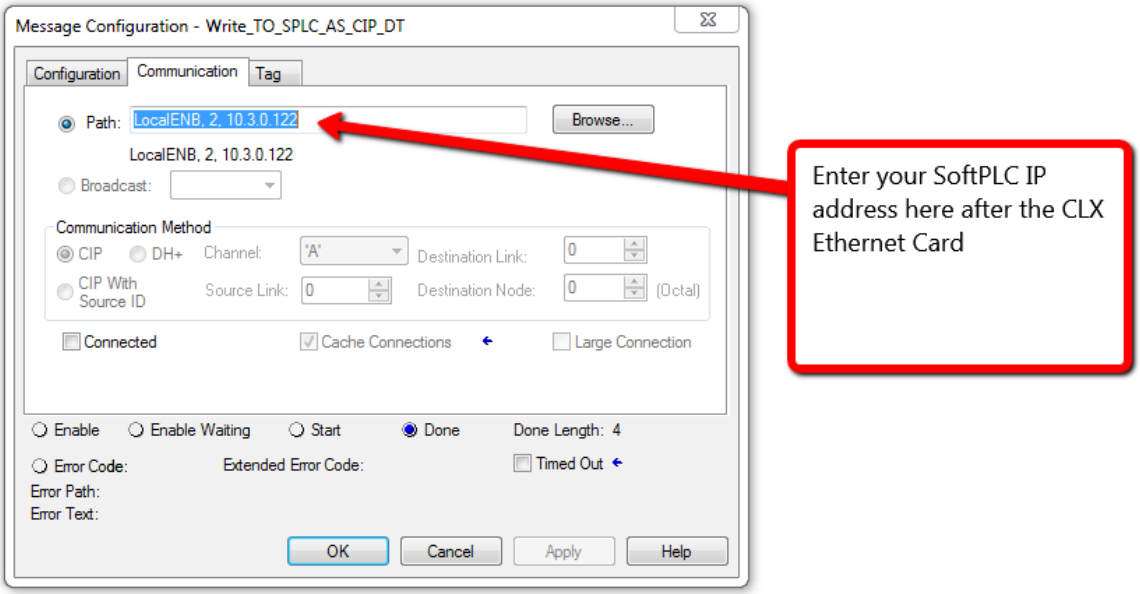


A.2.2. Write Data To SoftPLC via CIP Data Table Write

Enter the ladder logic shown in the figure below. This will initiate a WRITE of the configured target_role explicit assembly to SoftPLC from the Logix PLC.

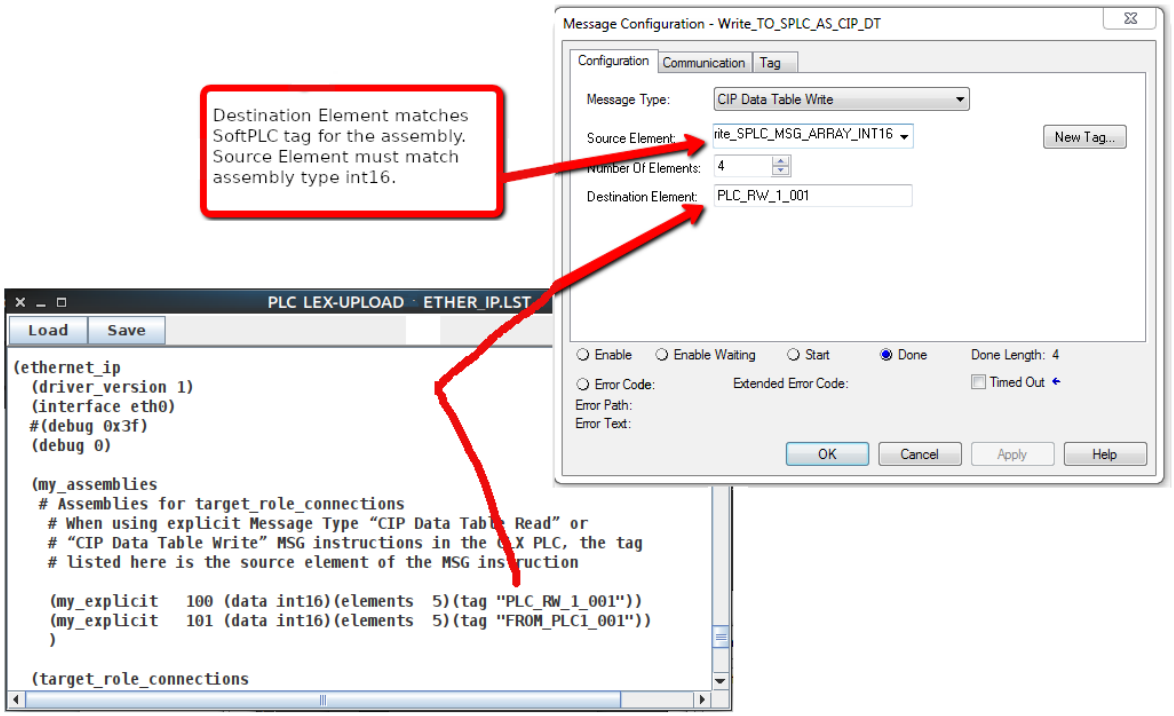


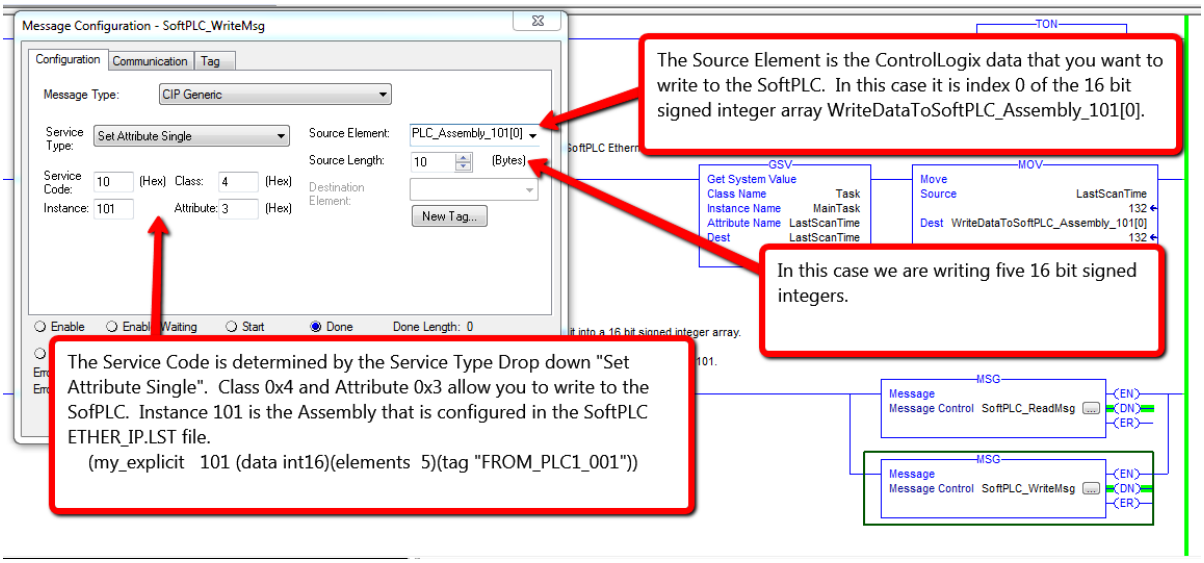
Configure the Message Instruction Path by entering the SoftPLC IP address after the Logix Ethernet interface ID.



In RSLogix, the Destination Element tag needs to match the tagname you will use in ETHER_IP.LST, and the Source Element INT array needs to be one the matches the SoftPLC int16 type.

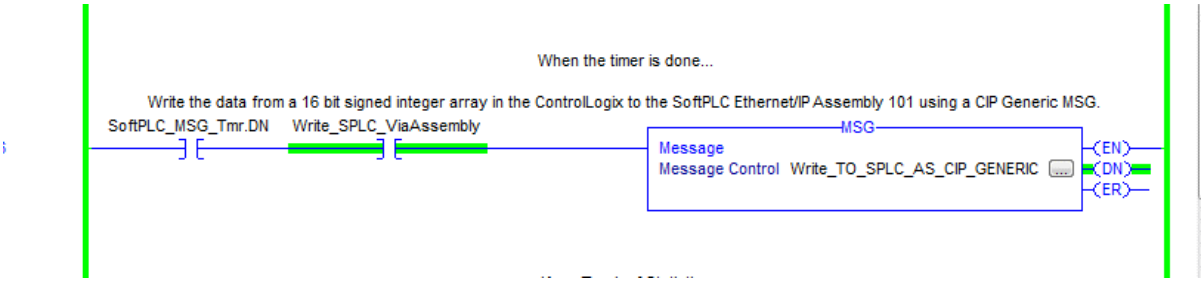
In the SoftPLC ETHER_IP.LST, configure the target assembly that supports the MSG instruction communications. This is an “explicit” type assembly, which is a bi-directional type assembly.



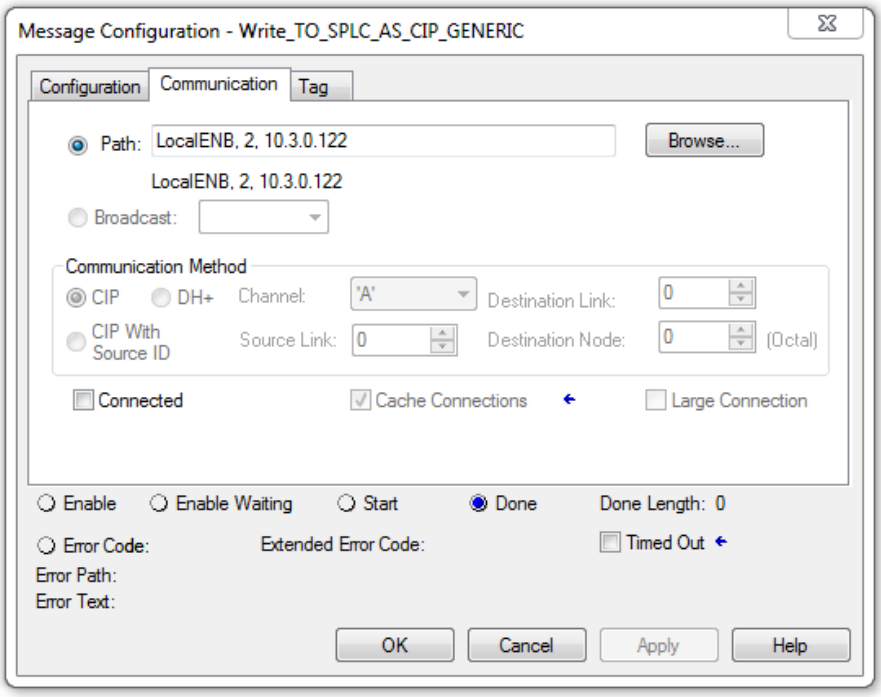


A.2.3. Write Data To SoftPLC via CIP Generic Message

Enter the ladder logic shown in the figure below. This will initiate a WRITE to SoftPLC from the Logix PLC using a CIP Generic Message.



Configure the Message Instruction Path by entering the SoftPLC IP address after the Logix Ethernet interface ID.



In RSLogix, the Instance needs to match the explicit assembly you will use in ETHER_IP.LST, and the

Source Element INT array needs to be one the matches the SoftPLC int16 type.

In the SoftPLC ETHER_IP.LST, configure the target assembly that supports the MSG instruction communications. This is an “explicit” type assembly, which is a bi-directional type assembly.